SafeNet
The Foundation of Information Security

Common Hacks and Counterattacks

A Guide to Protecting Software Products against Piracy

WHITE PAPER

OVERVIEW

In order to ensure they are properly paid for use of their application, Independent Software Vendors (ISV) must implement some type of software protection. The goal of any software protection solution is to restrict the use of software to abide by some specific license conditions. This can be done via hardware tokens, software-only licenses, or homegrown code.

External hardware-based solutions provide the highest level of security currently available. Unfortunately, like cockroaches, hackers are a persistent nuisance that cost software vendors worldwide billions of dollars in lost revenue. It is important to be sure that the hardware-based solution you select, as well as your implementation, seal off common hacker entry points. This white paper examines a variety of the most common hacking techniques, and the best counterattacks available for protecting your application from piracy.

GENERIC VS. SPECIFIC HACKS

Hacks on software protection dongles come in two forms: generic and specific. A generic hack means that the dongle itself is compromised. No amount of implementation improvement can counter a generic hack. All applications protected by the hacked dongle type are threatened. Specific hacks break only one specific dongle implementation for one particular software application. They do not pose a risk for other software vendors using the same dongle.

SECURE TUNNELING

In defending against attack, we must defend the communication between the token and the software application, as this can be the greatest point of potential vulnerability. In order to secure this communication, some hardware tokens use encryption algorithms to create a secure, hack-proof end to end tunnel.

In order to create a secure communication tunnel between the hardware token and the application, they must first exchange encryption keys. The key exchange process begins when the application generates a random AES (Advanced Encryption Standard) key. Generation of a new random key for every communication session greatly increases security. It is particularly important not to use a static key because that creates an increased vulnerability to attack.

The application then wraps the AES key using a public ECC (Elliptic Curve Cryptography) key. Next, the driver transfers the wrapped AES key to the token. The token, upon receipt, unwraps the AES key using its private ECC key and the key exchange process is complete. Storing the private ECC key in the token, rather than the application, also greatly increases security. Hacking a software application is relatively easier than hacking the token because it runs on a well understood hardware/OS platform for which plenty of third party tools exist to assist in debugging and reverse engineering programs. The same is not true for the firmware that runs inside the token and therefore debugging the token firmware to find the keys is not an easy undertaking for anyone other than the token manufacturer.

All communications between the driver and the dongle are now protected using the encryption key that has been exchanged. The key exchange process is a form of public key cryptography and results in the creation of a session based symmetric keys that protect the communication. The token and application communicate through a secure tunnel by encrypting and decrypting all communication using the AES key. For each subsequent communication session, a new AES key will be used.

The AES algorithm was adopted by National Institute of Standards and Technology (NIST) in November 2001 and is considered mathematically improbable to hack.

Elliptic Curve Cryptography (ECC) is stronger yet and considered mathematically impossible to hack.

HACK: BRUTE FORCE ATTACKS

A brute force attack exhaustively attempts every security combination until the secret being attacked is compromised. Brute force attacks can compromise the algorithms that are central to the security of a dongle. This affects all dongles for only a particular software vendor. It is not a generic hack. Using short passwords on some APIs can lead to an increased weakness against this hack.

COUNTERATTACK

Preventing a brute force attack can be accomplished when the secret data is large enough to make cracking all combinations virtually impossible. In order to protect the algorithms that are central to the security of a dongle, the dongle should have a secure security kernel. For example, the security kernel of a dongle can be a 128-bit AES key, to encrypt and protect the algorithms central to the dongle.

HACK: DEVICE EMULATION

Device emulation occurs when a piece of software emulates the hardware dongle. All of the secrets of the dongle, such as key values, are put into the emulating software. The software emulates the device and presents itself to the application as if it were the hardware dongle. Knowing the key values enables the software to do the same operations as the token.

Device emulation is a generic hack on the dongle and attacks the secrets of the dongle, typically via a brute force attack which, when successful, results in ultimately cracking the secret.

COUNTERATTACK

Device emulation can be prevented by securing the storage area on the dongle, making the dongle virtually impossible to compromise using a brute force attack. One such means for securing the storage area is to use a 128-bit AES key which is impossible to attack using brute force with today's computing power.

HACK: RECORD/PLAYBACK

In a record/playback attack, all information exchanged between the application and the dongle is recorded. A middleware (software application) is then created by the hacker to mimic the responses provided by the dongle.

When a record/playback attack has been conducted, the application has been compromised. The vendor must redesign their protection implementation in order to combat this type of attack. It is not a generic hack, but affects all copies of the application in which a record/playback attack exists.

COUNTERATTACK

Record/playback attacks can be prevented by encrypting and randomizing the communication between the application and the dongle. Use of a random 128-bit key to encrypt the communication between the application and the dongle is an effective means for preventing this hack.

Use of a static key, or hard-coded value in the application, results in weakness against record/playback attacks. If a dongle uses symmetric keys, when sharing a secret, it is necessary to leave a copy of it in the application. When encrypting within the application, it is necessary to leave the key there. Whereas with secure tunneling, the private key is kept in the token and the public in the application.

When the public key is embedded in the application, if the code were decompiled, the hacker has access only to a public, rather than private key. Discovery of a public key does not create a security vulnerability. However, if a dongle uses static symmetric based keys, decompiling the application can reveal the secret for communication. This is a potential entry point for a hacker infestation, as all communications can be understood by the hacker.

HACK: STEALING SECRETS

Stealing secrets occurs when a security password is obtained, providing unauthorized access to the dongle. Once access to the dongle is obtained, hackers are able to manipulate the application using the dongle.

COUNTERATTACK

Passwords for the dongle should never be communicated without being encrypted. Data that is being sent to the dongle should always be encrypted. Never sending passwords and other secrets "in the clear" will result in effective protection against this hack.

However, even if an encrypted channel is used to protect secrets, it is important not to use a static key. Use of a static key results in a higher degree of vulnerability for information in the keys to be extracted.

HACK: DEVICE SHARING

Device sharing refers to multiple PCs sharing one dongle without authorization to do so. One dongle can be used by several machines on the same network. While the application itself is not in jeopardy, the software vendor will not receive the anticipated revenues for use of multiple copies of their application. Use of a static key for encryption will result in greater susceptibility to this attack.

There are two potential scenarios under which device sharing can be done:

1. VMWare Session: Multiple virtual operating systems running on one hardware platform. Sharing a license in this environment allows multiple applications to be run concurrently in each VMWare session (instead of just one). In this scenario, each VMWare session uses its own instance of the driver to communicate to the dongle.
2. USB Sharing Hub: A USB hub that can be attached to multiple computers to share a single USB device. In this case, every computer attached to the USB hub thinks it has its own dongle. This can allow multiple computers to run an application even though only a single license exists.

COUNTERATTACK

Software vendors need to ensure that the quantity of computers or device drivers accessing the dongle does not exceed the number the customer has paid for. Secure tunneling of a dongle can safeguard against multiple devices or machines trying to access the token.

Every use of the application will open an instance of the driver. However, two or more instances of the driver cannot communicate with the dongle through a secure tunnel. A smart dongle driver will protect against these scenarios and enforce only the allowed number of licenses to be run from the dongle. Lack of a driver can increase your vulnerability to this attack.

HACK: HARDWARE CLONING

Hardware cloning occurs when hardware tokens that authorize the protected application are duplicated. To duplicate hardware token memory, the hacker first purchases tokens from the same token vendor used by the ISV. The hacker then clones the tokens by copying memory contents from the original token.

COUNTERATTACK

Software vendors can ensure they are protected against cloning by verifying that the token's memory is encrypted. One possible solution is encrypting memory by token-unique 128-bit AES key. This process should be an automatic part of the protection provided by your token manufacturer. Failure to encrypt user data in token memory should be avoided.

HACK: TIME TAMPERING

Time tampering involves rolling back the system clock to cheat a time-based license. Using this technique, a trial version, intended for a limited time only, could be used indefinitely.

COUNTERATTACK

One method for securing trials is to simply offer limited or crippled versions of the software. However if you would like to create time-based licenses in order to offer software subscriptions or offer complete time-limited trials, time-tampering is a risk that must be assessed and battled.

One method for securing time-based licenses is a real time clock built into a hardware key. The would-be hacker would have to crack into the key to alter the clock and would destroy the whole key in the process. Unfortunately, keys equipped with real time clocks are more costly and also require an on-board battery that will eventually run out of fuel.

Fortunately, there are simple, clever methods to battle time-tampering without increasing costs. If a hardware token is equipped with the ability to check and store the current system time, developers may use this information to prevent time tampering. If the token detects any significant changes in system time, the application can be programmed to disable or run in a restricted mode. This solution will both enable you to offer time-based licensing options as well as secure time-limited trials. Theft resulting from endless use of a “free, time-limited trial” is prevented without the added expense and hassle of real time clocks.

CONCLUSION

In addition to selecting the most secure dongle technology available, a secure implementation is vital. For best practices and specific guidelines to ensure secure licensing of your software, irrespective of your licensing technology, refer to SafeNet’s whitepaper and Webinar, “A Developer's Guide to Software Protection: Applying Modern Coding Techniques to Securing Software.”

About SafeNet, Inc.

SafeNet is a global leader in information security. Founded more than 20 years ago, the company provides complete security utilizing its encryption technologies to protect communications, intellectual property and digital identities, and offers a full spectrum of products including hardware, software, and chips. UBS, Nokia, Fujitsu, Hitachi, Bank of America, Adobe, Cisco Systems, Microsoft, Samsung, Texas Instruments, the U.S. Departments of Defense and Homeland Security, the U.S. Internal Revenue Service and scores of other customers entrust their security needs to SafeNet. In 2007, SafeNet was taken private by Vector Capital.

For more information about SafeNet’s solutions for software protection and licensing, please visit www.safenet-inc.com/sentinel.